Collaborative Schema Matching Reconciliation

Nguyen Quoc Viet Hung[1], Xuan Hoai Luong[1], Zoltán Miklós[2], Tho Thanh Quan[3], and Karl Aberer[1]

[1] École Polytechnique Fédérale de Lausanne
{quocviethung.nguyen,xuan.luong,karl.aberer}@epfl.ch
[2] Université de Rennes 1
{zoltan.miklos}@univ-rennes1.fr
[3] Ho Chi Minh City University of Technology
{qttho}@cse.hcmut.edu.vn

Abstract. Schema matching is the process of establishing correspondences between the attributes of database schemas for data integration purpose. Although several schema matching tools have been developed, their results are often incomplete or erroneous. To obtain correct attribute correspondences, in practice, human experts edit the mapping results and fix the mapping problems. As the scale and complexity of data integration tasks have increased dramatically in recent years, the reconciliation phase becomes more and more a bottleneck. Moreover, one often needs to establish the correspondences in not only between two but a network of schemas simultaneously. In such reconciliation settings, it is desirable to involve several experts. In this paper, we propose a tool that supports a group of experts to collaboratively reconcile a set of matched correspondences. The experts might have conflicting views whether a given correspondence is correct or not. As one expects global consistency conditions in the network, the conflict resolution might require discussion and negotiation among the experts to resolve such disagreements. We have developed techniques and a tool that allow approaching this reconciliation phase in a systematic way. We represent the expert's views as arguments to enable formal reasoning on the assertions of the experts. We detect complex dependencies in their arguments, guide and present them the possible consequences of their decisions. These techniques thus can greatly help them to overlook the complex cases and work more effectively.

1 Introduction

Integrating data stored in autonomously-developed information systems is essential for a number of applications. Schema matching is the process of establishing correspondences between the attributes of two database schemas, which is crucial for data integration. There is a large body of work on schema matching techniques: a number of commercial and academic tools (so called *matchers*) have been developed [4, 40]. Although some matchers perform impressively on certain data sets, the heuristic nature of the matching algorithms prevent them from yielding completely correct results. In practice, data integration tasks often include a post-matching phase, called *schema matching reconciliation*, where human experts review, validate and correct the generated correspondences. Although this phase requires substantial efforts, it has received very little attention, with the notable exception of [13].

In real life scenarios, the data to be integrated is often stored in several distributed databases. The creation of a globally accepted schema or ontology is neither practical nor possible for certain cases and one has to construct a set of pairwise mappings between the involved schemas. Constructing pairwise mappings can bring a number of advantages, e.g. if new schemas are added or removed, one only needs to adjust the local mappings. One can also construct the mappings in a pay-as-you-go fashion, whenever they are needed. The pairwise mapping establishment has also disadvantages. In this case, the matching reconciliation task can be very challenging as we explain in the following motivating scenario. Our work proposes methods to minimize these difficulties.

1.1 Motivating example

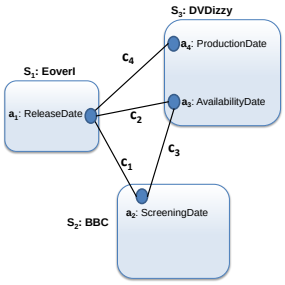
Let us consider the following scenario where three video content providers EoverI, BBC, and DVDizzy would like to create a shared website to publicize their offerings, which link back to the particular website for the purchases. The shared website needs information from the individual content providers (e.g. title, release date, main actors) so that consumers searching on the site can find the products they want. Although it would be conceivable to construct a global schema for the three providers, as more providers would join to this shared site, such a global schema could become impractical. We assume a scenario where the correspondences are established in a pairwise manner.

Figure 1a shows simplified schemas to illustrate this scenario. The three boxes represent the schemas of EoverI, BBC, and DVDizzy respectively. The figure shows four correspondences c_1, c_2, c_3, and c_4, generated by the matcher that we applied for each pair of schemas. As the names of the involved attributes are rather similar (ProductionDate, AvailabilityDate, ScreeningDate, ReleaseDate), typical matchers often fail to give the correct attribute mappings. Even human experts might disagree on which correspondences are correct. Moreover, as matchers only consider two schemas as input [4], they ignore natural expectations of the users with respect to the entire network. One can formulate these network-level consistency conditions as constraints:

- **One-to-one constraint**. In some cases, one expects that each attribute of one of the schemas is matched to at most one attribute of any other schemas.
- **Cycle constraint**. If the schemas are matched in a cycle, the matched attributes should form a closed cycle. This is a very natural requirement, if one would like to exchange data that is stored in the structure of the corresponding schemas.

Figure 1a shows some violations of these constraints, which are frequent in sets of automatically generated correspondences as most matchers do not take the constraints into account. For example, the attribute S_1.ReleaseDate is matched with two attributes S_3.ProductionDate and S_3.AvailabilityDate of schema S_3, thus violate the one-to-one constraint. The cycle constraint is violated also: S_3.AvailabilityDate is matched to S_2.ScreeningDate, which is then matched to S_1.ReleaseDate, and finishes at S_3.ProductionDate. Using such correspondences for data integration can lead to unwanted effects: AvailibilityDate and ProductionDate of DVDizzy would be mixed up.

If one would like to use the attribute correspondences for data integration or exchange, it is necessary to eliminate the errors from the set of correspondences.



(a)

	User Inputs	Arguments (derived from user inputs)
EoverI	approve c_1, c_2, c_3 disapprove c_4	$w_1^e = \langle\{c_1\}, c_1\rangle$, $w_2^e = \langle\{c_2\}, c_2\rangle$ $w_3^e = \langle\{c_3\}, c_3\rangle$, $w_4^e = \langle\{c_2, \neg c_2 \vee \neg c_4\}, \neg c_4\rangle$ $w_5^e = \langle\{\neg c_3 \vee \neg c_1 \vee \neg c_4, c_1, c_3\}, \neg c_4\rangle$ $w_6^e = \langle\{\neg c_4\}, \neg c_4\rangle$
BBC	approve c_3, c_4	$w_1^b = \langle\{c_3\}, c_3\rangle$, $w_2^b = \langle\{c_4\}, c_4\rangle$
DVDizzy	approve c_3, c_4	$w_1^d = \langle\{c_3\}, c_3\rangle$, $w_2^d = \langle\{c_4\}, c_4\rangle$

(b)

Fig. 1: The motivating example. (a) A network of schemas and correspondences generated by matchers. There are two violations: $\{c_2, c_4\}$ w.r.t. the *one-to-one* constraint, $\{c_1, c_3, c_4\}$ w.r.t. the *cycle* constraint. (b) An illustrated collaborative reconciliation between three video content providers: EoverI, BBC, and DVDizzy. The assertions (approvals/disapprovals) of BBC and DVDizzy are identical and different from those of EoverI.

This reconciliation process typically involves human input, which is a set of assertions that indicate whether a particular mapping, such as the correspondence between S_3.ProductionDate and S_1.ReleaseDate, should be approved or disapproved.

Until recently, this task was performed by a single expert. As the size of networks in data integration grows, the complex reconciliation tasks should be performed by not only one but several experts, to avoid the overload on a single expert and also to assign each expert the parts of the problem about which he is more familiar. In such cases, the experts might disagree about certain correspondences. If the application requires the network-level constraints, there could be rather complex dependencies among correspondences. For example, the choice of considering a correspondence *correct* can influence the possible choices for other correspondences. Moreover, the simple techniques for conflict resolution such as majority voting are not applicable, as the resulting set of correspondences would not comply to these constraints. To resolve these problems, the experts need to discuss and negotiate which correspondences to accept or reject. Because of complex dependencies in these networks, it is very challenging for the experts to overlook all possible consequences of their decisions. Thus on one hand it is highly desirable to split the reconciliation task, on the other hand combining individual results is very challenging. Our work addresses exactly this problem by proposing a number of services and a tool realizing those services to enable the collaborative process.

1.2 Contributions

In this paper, we leverage the theoretical advances and the negotiation nature of argumentation [5, 16] to support the work of multiple experts in the reconciliation task. This task, with the presence of consistency constraints, requires that the experts overlook a number of dependencies, which is very challenging without any support.

The specific contributions of our work are as follows. We model the schema matching network and the reconciliation process, where we relate the experts' assertions and the constraints of the matching network to an *argumentation framework* [16]. Our representation not only captures the experts' belief and their explanations, but also en-

ables to reason about these captured inputs. On top of this representation, we develop support techniques for experts to detect conflicts in a set of their assertions. Then we guide the conflict resolution by offering two primitives: *conflict-structure interpretation* and *what-if analysis*. While the former presents meaningful interpretations for the conflicts and various heuristic metrics, the latter can greatly help the experts to understand the consequences of their own decisions as well as those of others. Last but not least, we implement an argumentation-based negotiation support tool for schema matching (ArgSM) [32], which realizes our methods to help the experts in the collaborative task.

Our paper is organized as follows. Section 2 consists the formulation of schema matching network reconciliation problem and the overview of our solution. Next, Section 3 discusses how to construct an argumentation framework for schema matching networks, while Section 4 deals with our guiding methods for conflict resolution. After that, Section 5 describes implementation details. Then, we present our tool, ArgSM, in Section 6. Section 7 presents the related work and Section 8 concludes the paper.

2 Model

In this section, we focus on modeling the collaborative reconciliation in schema matching. Firstly, we introduce the elements of a schema matching network, which is a network of schemas that are connected through pairwise mappings between them. Then we describe the process of collaborative reconciliation to validate the mappings of this network, which are generated by automatic matchers.

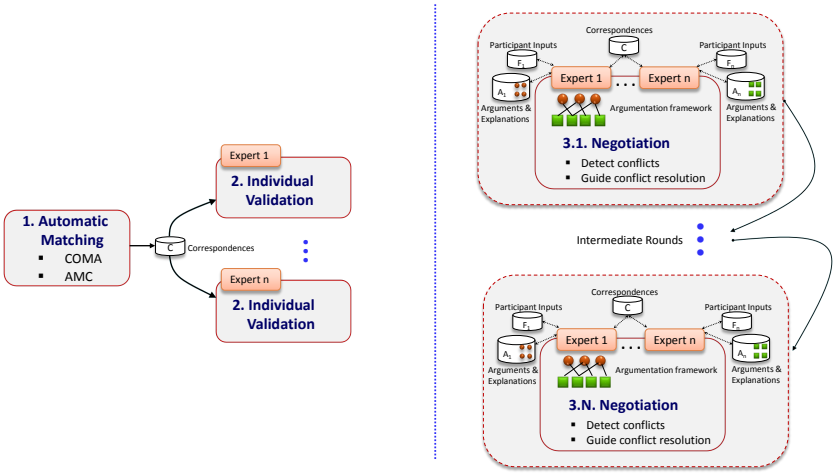
2.1 Schema Matching Network

We model a schema as a finite set of *attributes* $\{a_1, \cdots, a_n\}$. Let $\mathcal{S} = \{s_1, \cdots, s_n\}$ be the set of schemas of unique attributes ($s_i \cap s_j = \emptyset$ for all $1 \leq i, j \leq n$ and $i \neq j$) and $A_{\mathcal{S}}$ be the set of attributes in $\mathcal{S}$ ($A_{\mathcal{S}} = \bigcup_{i=1}^{n} s_i$). An *attribute correspondence* between a pair of schemas $s_1, s_2 \in \mathcal{S}$ is a pair of attributes (a, b) in which $a \in s_1$, $b \in s_2$. The set of all possible attribute correspondences is denoted by C. In addition, we have $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ as the set of integrity constraints, which expresses the intuitions observed from the schema matching problem. Combining the introduced notions, we define a *schema matching network* to be a triple $(\mathcal{S}, \Gamma, C)$.

In this paper, we consider a schema matching network $(\mathcal{S}, \Gamma, C)$, where C is typically the outcome of first-line schema matchers [20] such as COMA++ [1], AMC [37]. The schema matching reconciliation is the process of assessing the correspondences in C whether they are correct. The goal of the reconciliation process is to construct a maximal set of attribute correspondences ($M \subseteq C$) satisfying all constraints of Γ.

2.2 Collaborative Reconciliation

In practice, to obtain M, schema matching tasks often include a post-matching phase for correspondences to be reviewed and validated by experts. If the size of the schema matching network is large, the reconciliation task can be rather expensive. Moreover,



(a) Phase 1 - Individual validation **(b)** Phase 2 - Input combination

Fig. 2: The collaborative reconciliation process starts with a set of correspondences C generated by matchers. In Phase 1, each expert (user/participant) i is responsible for validating a particular set $C_i \subset C$. It is followed by Phase 2 that has multiple negotiation steps (3.1. to 3.N) to resolve conflicts in user inputs.

some experts might be more knowledgeable about some parts of the network, thus in these cases it is very natural to split the task among multiple experts.

The collaborative reconciliation as illustrated in Figure 2 is a two-phase process: *individual validation* and *input combination*. Let $\mathcal{E}$ denote the set of users $\mathcal{E} = \{E_1, \ldots, E_n\}$ who participate.

- *Individual validation:* In the first phase, each expert E_i is assigned to validate a subset C_i of C (usually of equal size). The assigned sets C_i usually overlap to a certain degree, thus there are correspondences assessed by several experts.
- *Input combination:* In the second phase, the individual inputs are combined. The goal of the collaborative reconciliation process is to construct a set of correspondences M that satisfies all constraints. If there are conflicting views about correspondences (for example, one expert considers correct while the other incorrect) then they need to come to a conclusion and chose which view to accept.

We leverage existing techniques from a large body of research [2, 27, 39] for the first phase. In this paper, we focus on the second phase. More precisely, we apply the theoretical advances of argumentation to detect conflicts in user inputs and guide them to resolve conflicts. Section 3, 4, and 5 will describe those functionalities in detail.

3 Argumentation for Schema Matching Networks

Let us consider a setting where several experts assess a set of attribute correspondences in a schema matching network. They might have different views whether a given correspondence should be correct or not. To complete the reconciliation task, they need

to discuss and resolve these conflicts to obtain a globally consistent set of correspondences. The conflicts between the different user views can be rather complex in the presence of integrity constraints. We call a situation *direct* conflict if two experts disagree about a given correspondence (one of them thinks it is correct, while the other claims that it is incorrect). In the presence of integrity constraints, we can also talk about *indirect* conflicts. For example, in Figure 1, if we assume the one-to-one constraint between S_1 and S_3 and an expert considers c_4 correct, then c_2 must be incorrect (otherwise the constraint would be violated). We call a situation where a second expert thinks that c_2 is correct an indirect conflict.

3.1 Arguments for schema matching

In order to study the conflicts between the experts opinions, we will rely on argumentation techniques. Argumentation is a systematic study of techniques to reach conclusions from given premises [5]. We will use the standard representation of arguments [5] where an *argument* consists of a *claim* and a *support* that explains the respective claim. We represent arguments in the form $\langle\{support\}, claim\rangle$. In the case of logic-based argumentation, both the support and the claim are logical formulas, such that the support is a minimal set that is sufficient to prove the claim.

For the schema matching problem, given a set of correspondences C, we employ propositional logic to encode the user inputs (during the reconciliation process): if an expert asserts that a given correspondence c is *true*, then we can represent this by a propositional variable v_c (and the assignment $v_c = true$). To simplify our notation, sometimes we use c to denote the propositional variable (corresponding to a correspondence c). This representation also enables to represent the consistency constraints, for example $\neg(c_2 \wedge c_4)$ encodes the *one-to-one* constraint (the two correspondences c_2 and c_4 cannot be true at the same time).

If several experts assess a set of correspondences (or as it is more common, they work on a different but overlapping set of correspondences) then we can use this encoding to represent their individual input in the form of arguments. For example, in Figure 1, we can represent the input assertion of an expert by the argument $w_1^e = \langle\{c_1\}, c_1\rangle$, where the claim is that the c_1 is correct, that is based on the simple support that is the knowledge of the expert about the correctness of c_1. A more complex example is the argument (that is the claim of an expert, together with a support) $w_4^e = \langle\{c_2, \neg c_2 \vee \neg c_4\}, \neg c_4\rangle$. This can be interpreted as follows: the expert has approved c_2, he would like to avoid violating the *one-to-one* constraint ($\neg c_2 \vee \neg c_4$), he disapproves c_4. We give a more complete list of arguments of three experts in Figure 1b (with respect to one-to-one and cycle constraints). In our work, the claim of an argument is always a single propositional clause, while the support is a set of propositional formulae.

3.2 Understanding the conflicting arguments

This representation enables us to explain more precisely the direct and indirect conflicts. If the claim of two arguments w_1 and w_2 contradict each other (together they form an inconsistent set of formulae) then we say that the arguments w_1 and w_2 are in direct conflict. In argumentation terminology this is called *rebuttal*. If the claim of an argument

w_2 appears in a negated form in the support of w_1, we talk about indirect conflict. In argumentation terms, w_1 *undercuts* w_2. In the following, we will consider following attack relation: w_1 *defeats* w_2 if w_1 either undercuts or rebuts w_2. For example, w_1^b attacks w_3^e and the argument w_1^d rebuts the argument w_5^e (Figure 1b). A set of arguments and attacks between the arguments $\langle A, R\rangle$ is called an argumentation framework [16].

Constructing argumentation framework $\langle A, R\rangle$ for a reconciliation problem from the arguments of all the experts $A = \bigcup A_i$ with the above attack relation has several advantages. In particular, computing the attack relation we can detect each problem that might exists in the experts' inputs. The argumentation framework enables even more complex tasks that we explain in the following sections.

4 Guiding the Conflict Resolution

We have presented in Section 3 how to construct an argumentation framework for the schema matching problem, where multiple experts work on the reconciliation task. This not only enables to reason about the user input, but also to detect conflicts and determine the reasons for these problems. In this section we focus on techniques that exploit this information to guide the experts in resolving these conflicts.

In particular, we describe here two services that can largely help the collaborating experts. These are the (1) the *interpretation of conflict structures*, with which we can present meaningful interpretations for the conflicts together with some associated metrics that can largely support the negotiation of the experts and the (2) *what-if analysis*, with which we can compute (and in the tool visualize) the consequences of a particular potential decision. The details of these services are provided in what follows.

4.1 Interpretation of conflict structures

Given a potentially conflicting set of assertions from several experts who collaborate on the reconciliation task, we can analyse the structure of conflicts and compute (qualitative) metrics that explain the conflicts as well the potential ways to resolve the problems.

Extensions. An *extension* $\epsilon \subseteq A$ of an argumentation framework $\langle A, R\rangle$is an acceptable set of arguments ϵ; i.e., a set of arguments that can be accepted simultaneously, that is, they do not contain any conflicts. In fact, there are many possible ways to define an extension. The various strategies that can be used to construct such an exception are called *acceptability semantics* [16]. For instance, an extension following the *complete* semantics is a conflict-free set of arguments, defends all its arguments, and contains all arguments it defends. A set of arguments A' is conflict-free if there are no arguments $a_1, a_2 \in A'$ that a_1 attacks a_2. Meanwhile, a_1 defends a_2 if there exists a_3 that is attacked by a_1 and attacks a_2. Generally, there are more then one extension (for a given semantics). Hence, the experts need to agree about choosing which one. With argumentation, we can compute and present extensions for the experts, thus enables them to consider all possible options.

Witnesses. Given a set of conflicting arguments, we compute (also present and visualize) witnesses of the conflicts. Let A_c and $A_{\neg c}$ be the set of arguments having claim c

and $\neg c$ respectively, i.e. $A_c = \{(\Phi, \alpha) \in A \mid \alpha = c\}$ and $A_{\neg c} = \{(\Phi, \alpha) \in A \mid \alpha = \neg c\}$. A_c and $A_{\neg c}$ explain why we should approve/ disapprove c. Presenting the witnesses lets the experts understand problems arise during reconciliation.

Example 1 *In Figure 1, EoverI wants to approve c_1, c_2, c_3 and disapprove c_4. From the complete semantics, we obtain the extension $\{w_1^e, w_3^e, w_1^b, w_4^e, w_5^e, w_6^e\}$. Based on the explanations provided by this extension's arguments, EoverI can better argue for the other two participants to approve c_1 (explained by w_1^e), c_2 (explained by w_2^e), and disapprove c_4 (explained by w_4^e, w_5^e, and w_6^e). Furthermore, observing the witnesses for and against c_4 would make the decision to discard this correspondence more convincing. Indeed, from those witnesses ($A_{c_4} = \{w_2^b\}$ and $A_{\neg c_4} = \{w_4^e, w_5^e, w_6^e\}$), we realize that although the decision supporting c_4 is voted by more users, the opposite decision (to disapprove c_4) seems to be the one that is better explained. Moreover, disapproving c_4 is also justified by intuitive observations on the network: the approval of this correspondence would constitute not only a one-to-one constraint violation (with c_2) but also a cycle constraint violation (with c_1 and c_3).*

We cannot only compute the extensions and witnesses to facilitate the discussion among the experts, but also associate heuristic metrics to further support their work. Furthermore, we enable the users to rank decisions, based on the strengths of their explanations (arguments) or the decisions themselves.

Argument strength. Computing extensions is a preliminary step to evaluate arguments. From the occurrences of an argument in the extensions, we compute the *argument strength*. Given the set of extensions $\mathcal{E}$ of an argumentation framework with respect to an acceptability semantics, the strength of an argument a is the number of its occurrences in $\mathcal{E}$ divided by the size of $\mathcal{E}$:

$$argument\_strength(a) = \frac{\sum_{\epsilon \in \mathcal{E}} \mathbb{1}_{a \in \epsilon}}{\mid \mathcal{E} \mid}$$

With *argument strength*, we have a more fine-grained metric to rank arguments and assist the users to make wiser decisions. Indeed, we were motivated by the notion of *argument acceptance* [12], which evaluates arguments based on their occurrences in all extensions of a given acceptability semantics. However, this is a rough metric, which does not take into account the difference between the number of occurrences of arguments in the extensions. This shortcoming prevents the users from having detailed looks on the credibility of arguments to compare them.

Decision strength. Providing explanations might be overwhelming for the users, especially when there are too many arguments. Therefore, we associate each decision with a quantitative metric reflecting the *decision strength*, which is computed by applying aggregate operators (max, min, avg, etc.) on the set of supporting arguments. Based on this metric, the users can evaluate which decisions should be made given a specific circumstance. It is also important to note that the number of ambiguous correspondences is generally large. As a result, identifying the sequence of correspondences to negotiate is necessary. In practice, one may define such a sequence by taking pairs of decisions for and against a correspondence in the ascending order of the level of ambiguity, which can be measured by the difference between the strengths of associated decisions.

Example 2 *An example of argument and decision strength can be found in Figure 3. In that figure, we have on the left the decisions (circle shapes) supporting and opposing each correspondence in the network, as well as the associated arguments (square shapes). We follow the complete acceptability semantics to compute argument strengths and apply the* sum *operator to evaluate decision strengths. Those values are displayed right above the corresponding shapes. We can observe that the decision to disapprove* c_4 *possesses higher strength. Thus, disapproving* c_4 *would be the better decision to take. In fact, this outcome aligns with what the users should achieve using the qualitative metrics solely. Having the quantitative metrics (argument and decision strength), however, brings the users more details thus increases the confidence in making decisions.*

4.2 What-if analysis

We have also developed another reasoning service, called *what-if analysis*, that exists in many decision support systems [25] and largely supports the collaborative work. This service can compute (and in our tool also visualize) the consequences of a possible decision. Using the what-if analysis, the experts can understand the consequences of any particular decision, resulting in a stronger feeling of trust throughout their work. Three questions for which we can compute the answers are:

- Q_1: *Which* ***arguments*** *will be added or deleted if a particular assertion is given?*
- Q_2: *Which* ***attacks*** *will be added or deleted if a particular assertion is given?*
- Q_3: *Which* ***extensions*** *will be modified if a particular assertion is given?*

By answering these questions, we can provide the experts with two important views. The (1) *Local view* reflects the relationships among inputs of a single participant. Each participating expert can check whether his new assertion conflicts with the previous inputs. Technically, we use the answers of Q_1 and Q_2 to construct this view. When an user gives a new assertion, his own arguments are maintained. If any attack between those arguments is found, the user is notified to adjust his inputs to avoid further inconsistencies. Besides, the (2) *Global view* reflects the connections between inputs of multiple users. All participants can observe the negotiation progress. To construct this view, we use the answers of Q_2 and Q_3. The number of attacks and extensions are maintained. In one hand, the users understand the current conflicts (attacks) among their arguments and the impact of those inconsistencies. In the other hand, keeping track of the extensions lets them know the current state of the system and when it reaches an agreement.

Example 3 *In Figure 1, three video content providers now attempt to change their assertions to reach an agreement. In the view-point of EoverI, he might change his disapproval of* c_4 *since the others both approve* c_4*. If EoverI approves* c_4*, two new arguments* $w_7^e = \{\{c_4\}, c_4\}$*,* $w_8^e = \{\{c_4, \neg c_2 \vee \neg c_4\}, \neg c_2\}$ *and two new attacks* $w_7^e \leftrightarrow w_4^e$*,* $w_8^e \leftrightarrow w_2^e$ *will be added. Through local view, EoverI can foresee these new arguments and attacks to realize the contradiction with himself. In the view-point of BBC and DVDizzy, they might change the approval of* c_4 *because of EoverI. If they disapprove* c_4*, two arguments* w_2^d *and* w_2^b *will be deleted; and hence, there is no attack between remaining arguments and only one extension remains. Through global view, they can foresee this consequence and feel more confident to make changes. In addition, they might also agree with EoverI on* c_1 *and* c_2 *since no further contradiction exists.*

5 Implementation

In the previous sections, we discussed how to support the collaborative reconciliation through detecting conflicts in the assertions of multiple experts and guiding the resolution of these conflicts. To accomplish these tasks, we need to realize the argumentation framework, which can only be achieved after generating arguments and computing the attack relation. This section serves a two-fold purpose. First, we present how to instantiate an argumentation framework using ASP-based tools. Second, we describe how to implement the proposed services on top of this argumentation framework.

5.1 Instantiate Argumentation Framework

We rely on a declarative language, Answer Set Programming (ASP) [7], to carry out the preliminary tasks before instantiating an argumentation framework (Figure 2). In our work, we utilize the ASP Solver DLV-Complex[8] to take advantages of its built-in data structures and functions. We invoke an ASP program called Π_{pre}, which is essentially the union of other ASP program, each of which is responsible for a specific task. In particular, $\Pi_{pre} = \Pi_{smn} \cup \Pi_{\Gamma} \cup \Pi_{inputs} \cup \Pi_{\Phi}$.

Encoding the schema matching network (Π_{smn}). We extend the setting to keep track of the correspondences. Let I be the set of identities, a function $f : C \rightarrow I$ maps exactly one element in I to each in C. f is *injective* as an identity is assigned to at most one correspondence. For each schema $s_i \in \mathcal{S}$, we represent the attribute-schema relationship between s_i and each attribute $a \in A_{s_i}$ as a ground fact $attr(a, s_i)$. It is assumed that the attributes are globally unique. Meanwhile, the correspondence-attribute relationships are captured by $cor(c, a_i, a_j)$ for each $(a_i, a_j) \in C$ and $c = f((a_1, a_2))$. For instance, the network in Figure 1 has the following encoding of Π_{smn}:

$$\begin{aligned}\Pi_{smn} = \{&attr(a_1, S_1).attr(a_2, S_2).attr(a_3, S_3).attr(a_4, S_3).\\ &cor(c_1, a_1, a_2).cor(c_2, a_1, a_3).cor(c_3, a_2, a_3).cor(c_4, a_1, a_4).\}\end{aligned}$$

Encoding the integrity constraints (Π_{Γ}). We encode the integrity constraints as rules. In fact, we use rules to encode two different types of constraints. The first type is our basic assumptions, which are encoded in the program π_{ass} below:

$$\begin{aligned}\pi_{ass} = \{ &\leftarrow attr(a, s), attr(a, s'), s \neq s'.\\ &\leftarrow cor(c, a, a'), attr(a, s), attr(a', s'), s = s.'\}\end{aligned} \tag{1}$$

Second, we use rules to express the network-level integrity constraints. Atoms prefixed with # are built-in functions of DLV-Complex.

– *Cycle constraint*: a path of correspondences must not make any two attributes of a schema reachable. Each $rch(S, a, a')$ signifies the reachability between attributes a and a' via the correspondences in S. Below is the encoding of the program π_{cycle}:

$$\begin{aligned}\pi_{cycle} = \{&rch(S, a, a') \leftarrow cor(c, a, a'), \#set(c, S).\\ &rch(S, a, a') \leftarrow \#intersection(P, Q, R), \#card(R, 0),\\ &rch(P, a, b), rch(Q, b, a'), \#union(P, Q, S).\\ &violation(S) \leftarrow rch(S, a, b), a \neq b, attr(a, s), attr(b, s).\}\end{aligned} \tag{2}$$

– *One-to-one constraint*: any attribute is matched by at most one attribute in each of the other schemas. Each $pair(c, c\prime)$ captures a violation detected by π_{1-1} below:

$$\begin{aligned}\pi_{1-1} = \{&pair(c, c') \leftarrow cor(c, a, b), cor(c', a, b'), b \neq b', attr(b, s), attr(b', s).\\ &violation(S) \leftarrow pair(c, c'), \#set(c, c', S).\}\end{aligned} \tag{3}$$

To put it in a nutshell, Π_Γ is defined formally as $\Pi_\Gamma = \pi_{ass} \cup \pi_{cycle} \cup \pi_{1-1}$. For example, invoking Π_Γ for the network in Figure 1, we have π_{ass} check the validity of the schema matching network encoded by Π_{smn}, which is valid indeed. Besides, from π_{cycle} and π_{1-1}, we obtain two violations: $\{c_2, c_4\}$ and $\{c_1, c_3, c_4\}$.

Collecting user inputs (Π_{inputs}). Indeed, user inputs are *assertions* on the correspondences. We represent user assertions (that may be *approvals* or *disapprovals* of correspondences) as ground atoms of the form $app(\cdot)$ and $dis(\cdot)$. For instance, in Figure 1b, the user EoverI approves c_1, c_2, c_3 and disapproves c_4, while BBC and DVDizzy only approve c_3 and c_4. Their inputs are encoded as follows:

$$\Pi^{EoverI}_{inputs} = \{app(c_1).\ app(c_2).\ app(c_3).\ dis(c_4).\}$$
$$\Pi^{BBC}_{inputs} = \Pi^{DVDizzy}_{inputs} = \{app(c_3).\ app(c_4).\}$$

Constructing the set of formulae (Π_Φ). We construct this set by extracting propositional formulae from either the assertions (through π_{simple}) or the detected violations (through $\pi_{extract}$). Formally, $\Pi_\Phi = \pi_{simple} \cup \pi_{extract}$. Each atom $kb(\cdot)$ captures a formula. For assertions, we consider each assertion $app(c)$ (or $(dis(c))$ as a simple formula c (or $\neg c$). This is capture by the program π_{simple}:

$$\begin{aligned}\pi_{simple} = \{&kb(c) \leftarrow app(c).\\ &kb(\mathsf{neg}(c)) \leftarrow dis(c).\}\end{aligned} \tag{4}$$

Things are more complex in the cases of the constraint violations (detected by the π_{cycle} and π_{1-1} of Π_Γ). From a violation $\{c_1, \cdots, c_n\}$, we can state that at least one of the assertions must be false. This is expressed formally by the formula $\neg c_1 \vee \cdots \vee \neg c_n$. Such formulae are extracted from the detected violations by the program $\pi_{extract}$, whose process is described below:

– Initially, violations are captured by ground atoms $violation(S)$ where S is a set of correspondences. We convert from set to list using the atom by $vioList(L)$, in which L is the list-based representation of S.
– From each list L, we create sublists starting from the first to the $(n-1)$th element $([c_1, \cdots, c_n], [c_2, \cdots, c_n], \cdots, [c_{n-1}, c_n])$.
– Based on the sublest, we form formulae in a bottom-up manner, starting from the shortest one $([c_{n-1}, c_n])$. The longer sublists have their forumlae composed recursively from those that are one element shorter. This is continued up to original list.

Example 4 *In Figure 1b, we have the collected the inputs of EoverI. Through π_{simple}, we obtain the formulae $kb(c_1)$, $kb(c_2)$, $kb(c_3)$, and $kb(\mathsf{neg}(c_4))$. Besides, we also detected the violations in the schema matching network, from which $\pi_{extract}$ form two formulae $kb(\mathsf{or}(\mathsf{neg}(c_2), \mathsf{neg}(c_4)))$ and $kb(\mathsf{or}(\mathsf{neg}(c_1), \mathsf{or}(\mathsf{neg}(c_3), \mathsf{neg}(c_4))))$. These ground atoms $kb(\cdot)$ compose the set of formulae of EoverI.*

With the set of formula, we proceed to first generate arguments then the attack relation, with the goal to compose an argumentation framework. One could consider formulae in the set we just obtained as candidates to be argument claims. This approach,

however, would easily overwhelm the users due to the huge amount of generated arguments. The reason is that many formulae are syntactically different but semantically equivalent. To avoid this scenario, we limit the candidates for argument claims. In practice, users concern more with arguments claiming to approve or disapprove correspondences. We thus select the set of possible claims from the assertions. In the motivating example, the possible claims for EoverI is $cl(c_1)$, $cl(c_2)$, $cl(c_3)$, and $cl(\mathsf{neg}(c_4))$.

We take advantage of Vispartix [11], an ASP-based tool, to not only generate arguments but also to compute the attack relation. For argument generation, the tool considers only subsets of the set of formulae and the set of possible claims. It then looks for pairs which can be considered as arguments (Figure 1b presents an example of these generated arguments). Once the set of arguments is ready, we start to compute the attack relation. This is done by invoking the corresponding feature of Vispartix with the set of all arguments (the union of the arguments of each user) as the input. Visaprtix provides the users with several attack types [5], such as *defeat*, *undercut*, and *rebut*.

5.2 Realizing Services

In Section 3 and 4, we showed the elements of an argumentation framework as well as offered possible services (conflict detection, interpretation of conflict structures, what-if analysis) on top of this framework. In this subsection, we will describe how to realize these services, with the focus on technical aspects.

Conflict detection. We detect conflicts based on the results of ASP-solver in section 5.1. In that section, we described how to encode the integrity constraints in the language of ASP. The solver DLV-Complex is responsible for detecting the violations based on our encodings. Based on the results of the solver, we have the atoms $vioList(L)$ as lists of violation, each of which contains a set of involved correspondences. Moreover, in our system, we show not only the violations but also the explanations for these violations. In doing so, we analyze the attack relations R of the argumentation framework. The user inputs are valid if this *vioList* is empty or R is empty.

Interpretation of conflict structures. To realize this service, we need to compute four elements: the extension, the witness, the argument strength, and the decision strength. In Section 5.1, we already generated a set of arguments. As previously defined, a witness of a claim is a set of arguments having this claim. By grouping arguments sharing the same claim, we obtain the witnesses for all possible claims. Then, we employ Vispatrix [11] to generate all possible extensions with different semantics. After obtaining all extensions, we compute argument and decision strength as mentioned in Section 4.1.

What-if analysis. To realize this service, we need to recompute the argumentation framework and all possible extensions when user modifies an assertion. Then, we compare the differences between the new computed results and the current ones. Based on these differences, we can know what arguments, attacks, and extensions are added or deleted to answer three what-if questions in Section 4.2. This service is implemented with the support of Vispatrix [11], which allows efficient recomputation.

All above-mentioned services are integrated in our argumentation-based negotiation support tool, namely ArgSM. This tool not only implements these services but also provides graphical user interface. The details will be described in the next section.

6 Tool - ArgSM

ArgSM is an argumentation-based negotiation support tool for schema matching. It is developed by using the Java programming language and the JUNG[4] library for visualization purposes. We also integrate other supporting libraries, including Vispatrix [11] and DLV-Complex [8]. We have also made the source code to be publicly available at our website [5]. In this section, we discuss the user interface and the technical challenges.

6.1 User Interface

ArgSM can aggregate the assertions of multiple experts and visualize the concerned arguments. For *visualizing*, we provide users with a GUI (Figure 3), which is a unified view of the provided inputs and the argumentation framework. From the GUI, the users can compare their inputs via *views* and *view modes*. In particular, views give the users static pictures about the network, the decisions, and the explanations, while view modes provide the users with dynamic interaction during collaborative reconciliation.

Two views are supported in ArgSM: *Schema* and *Argumentation* view. They are displayed alongside each other in the GUI (Figure 3, from right to left). Together, they should help the users to review the inputs and make decisions effectively.

- **Schema view**. This view shows the schema matching network for the users who do not have deep understandings of argumentation, hence another name *User view*. In this view, correspondences are highlighted according on to their status. There are three possible status: (1) *all approved* and (2) *all disapproved* respectively for correspondences that are approved and disapproved by all users, and (3) *ambiguous* for those that are approved by some and disapproved by the others.
- **Argumentation view**. Also called the technical view, it is intended for those who have knowledge on argumentation. In this view, the numbers outside the shapes indicate the strengths of decisions (circles) or witnesses (squares) respectively. There are two perspectives supported:
  - *Decision-making perspective* shows all possible decisions (aggregated from the inputs) and the associated witnesses (arguments) that explain the reasons for making decisions.
  - *Abstract argumentation perspective* presents the argumentation framework in the form of a directed graph. The nodes are the arguments and the directed edges are elements of the attack relation.

Those views are further supported by three *view modes*. Apart from the *Normal mode*, which is set by default and has no interaction at all, the others allow users to interact with the network and the arguments:

- **Schema-Argumentation mode**. Upon clicking on a correspondence in the *Schema view*, the user can see all generated decisions (circle shapes) and witnesses (square shapes) in the *Argumentation view*. For instance, in Figure 1, correspondence c_3 has

[4] JUNG - `http://jung.sourceforge.net`
[5] `https://code.google.com/p/argsm/wiki/ArgSM`

two arguments w_4^e and w_5^e for disapproving and w_2^b for approving. Therefore, the users will have two decisions at their disposal (c_4 and $\neg c_4$, presented in circles) and three witnesses (w_4^e, w_5^e, and w_2^b, presented in squares). Those circles and squares will be highlighted when the users click on c_4 in the *Schema view*.

– **Argumentation-Schema mode**. There are two cases. First, when choosing a witness in the *Argumentation view*, the participants will see all the involved correspondences in the *Schema view*. Correspondences appearing in the support are showed differently from those in the claim of the witness. In the other case, once a decisions is clicked on, the relating correspondence is highlighted in the *Schema view*.

For a better understanding and stronger feelings of trust, ArgSM not only generates explanations but also provides the foreseeable effects of each decision. Technically, we keep the strength of arguments and the possible decisions up-to-date during negotiation.

6.2 Technical Challenges

When implementing ArgSM, we had to cope with a number of scalability issues. The schemas are usually too large, leading to high response time (i.e. computation time) for each human interaction and overwhelming control for the experts. To overcome such challenges, we apply the following techniques:

– **Partitioning**. We divide the correspondences into small disjoint and independent subsets such that any two correspondences in one subset share a common attribute.
– **Caching**. We apply the view maintenance technique [6] with a repository storing intermediate results along the process. The rationale behind is that collaborative reconciliation is incremental as a change (insertion or removal) only affects some arguments. Recomputing all arguments after each modification is unnecessary.
– **Filtering**. It is not useful to generate any and every argument. We only filter for arguments of predefined claims. Hence, not only does it reduce computation time but also avoid overwhelming the users. Every argumentation process should operate on the filtered set. That set may be refined by modifying the predefined claims.

To show the efficiency of the above techniques, we set up an experiment to measure the response time of computing arguments and attacks for a large network. With the help of automatic matchers, we obtain 472 correspondences in the network. Since the network is large, it is partitioned into 21 clusters, in which smallest and biggest ones contain 6 and 59 correspondences respectively. Applying caching and filtering for each cluster, the response time varies from 0.38s (the smallest cluster) to 12.92s (the biggest cluster). In total, it takes about 61.16s to generate all arguments and attacks.

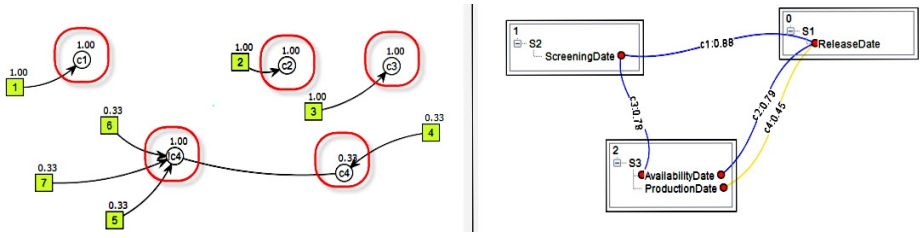


Fig. 3: The GUI of ArgSM, with *Argumentation view* (left) and *Schema view* (right)

7 Related Work

In this section, we first review some techniques and applications related to schema matching. We then review salient work that supports negotiation in collaborative information systems as well as applications built on top of argumentation-based negotiation.

7.1 Schema Matching - Research and Applications

Database schema matching is an active research field. The developments of this area have been summarized in two surveys [4, 40]. Existing works on schema matching focused mainly on improving quality parameters of matchers, such as precision or recall of the generated matchings. Recently, however, one started to realize that the extent to what precision and recall can be improved may be limited for general-purpose matching algorithms. Instead of designing new algorithms, there has been a shift towards matching combination and tuning methods. These works include YAM [14], top-k matchings generation [23], reuse of existing matchings [22], systematic matching ensemble selection [21], automatic tuning of the matcher parameters [30], or validation of correspondences with the help of schemas belonging to the same domain [44]. While there is a large body of works on schema matching, the post-matching reconciliation proces, which is central to our work, has received little attention in the literature. Recently, there are some works [13, 27, 33, 34] using pay-as-you-go integration method to establish the initial matching and then incrementally improve matching quality. While [13, 27, 33] depend on one user only, the framework in [34] relies on the work of multiple participants of the crowdsourcing community, who -unlike in our setting- do not communicate. The authors [34] propose constraint-based aggregation techniques for conflicting user inputs. The major difference between our work and [34] is that we assume the participants are domain experts on the field.

Regarding applications, one of the major challenges posed by the Intertnet and collaborative business is enterprise interoperability (EI) in recent years is the ability of two or more systems to exchange information with each other. Several relevant initiatives have been undertaken and are ongoing today worldwide. Among them, schema matching is an important primitive to tackle strategic challenges such as Interoperability Service Utility [17], Web Technologies for Enterprise Interoperability [31], Knowledge-Oriented Collaboration [35], and Science Base for Enterprise Interoperability [10]. The roles of schema matching in enterprise systems are summarized in [47].

7.2 Argumentation in Collaborative Work

In the literature, many researchers have studied collaborative work which involves multi participants with different and possibly conflicting interests. In order to resolve conflicts and reach mutually acceptable agreements, participants negotiate with each other. There is a large body of mechanisms dedicated to support this negotiation process, including game-theoretical approach [29, 45], heuristic-based approach [19, 28], and argumentation-based approach [36, 46]. The work of this paper focuses on the argumentation-based approach, which provides not only explanations for each decision but also a language for communication between participants [42, 43].

The argumentation-based approach has been successfully applied to many practical applications. In e-commerce systems [3], argumentation is used for solving conflicts that may arise among distributed providers in large scale networks of web services and resources, thus improving the automation level of business processes. In collaborative & cooperative planning [18], argumentation can be combined with other techniques (e.g. machine learning) to help participants collaborate to solve problems by determining what policies are operating by each participant. In social-network platforms [24], arguments can be extracted from natural language and then argumentation is used to determined the social agreements among participants. In cloud-computing [26], argumentation can be used to help cloud providers, who manage computational resources in the platform, to reach an agreement on reacting against physical failures. In semantic web [41], argumentation has been modeled under Argument Interchange Format (AIF) ontology, which forms the foundation for a large-scale collection of interconnected arguments in the Web. In this paper, we apply argumentation in data integration domain, which is an active research field for more than ten years [4].

From the argumentation point of view, our work is related to two main directions in argumentation research: *abstract* and *logical* argumentation. This classification can be found in [38], along with a discussion on the development of argumentation research. In brief, abstract argumentation was proposed in [16]. In that paper, the author used an *argumentation framework* to describe a system of arguments and attacks, which are actually considered as abstract objects, hence the name *abstract argumentation*. Acceptability semantics of arguments were studied in [16], [15], and [9]. To make abstract argumentation more applicable, there are attempts to give concrete definitions to arguments and attacks. The most prominent proposal is *logical argumentation* [5], which relies on propositional logic. In particular, an argument is a pair of *support* and *claim*, while attacks are defined on the logical inconsistencies between the supports and/or claims of arguments.

8 Conclusion

We presented an argumentation-based tool to support collaborative reconciliation, where multiple users, with different sorts of opinions, cooperate to validate the outputs of automatic matchers. While splitting the reconciliation task is highly desirable, combining the individual results in the presence of consistency constraints is very challenging for the collaborating experts. Our tool and its services shall facilitate collaboration. In particular, we systematically detect conflicts, provide the experts with visual information to understand the causes of the problems. Moreover, we offer services to better understand decision consequences and make collaborative reconciliation more transparent.

Our work opens up some future research directions. First, we will design a negotiation protocol to enable negotiation within our tool. Second, we would like to extend the notion of proposed constraints and consider further integrity constraints that are relevant in the praxis (e.g., functional dependencies, domain-specific constraints). Third, we would like to apply our methods to other problems. While our work focuses on schema matching, our techniques, especially the argumentation-based reconciliation, could be applicable to other tasks such as entity resolution or business process matching.

Acknowledgment

This research has received funding from the NisB project - European Union's Seventh Framework Programme (grant agreement number 256955) and the PlanetData project - Network of Excellence (grant agreement number 257641). Many thanks to SAP [6] and MOM [7] for providing use cases and data sets as well as evaluating our tool.

References

[1] D. Aumueller et al. "Schema and ontology matching with COMA++". In: *SIGMOD*. 2005, pp. 906–908.
[2] K Belhajjame. "User feedback as a first class citizen in information integration systems". In: *CIDR*. 2011, pp. 175–183.
[3] J. Bentahar et al. "Using argumentation to model and deploy agent-based B2B applications". In: *KBS* (2010), pp. 677–692.
[4] P. A. Bernstein, J. Madhavan, and E. Rahm. "Generic Schema Matching, Ten Years Later". In: *PVLDB* 4.11 (2011), pp. 695–701.
[5] P. Besnard and A. Hunter. *Elements of Argumentation*. The MIT Press, 2008.
[6] J. A. Blakeley, P.-A. Larson, and F. W. Tompa. "Efficiently updating materialized views". In: *SIGMOD Rec.* (1986), pp. 61–71.
[7] G. Brewka, T. Eiter, and M. Truszczyński. "Answer set programming at a glance". In: *Commun. ACM* (2011), pp. 92–103.
[8] F. Calimeri et al. "Computable Functions in ASP: Theory and Implementation". In: *ICLP*. 2008, pp. 407–424.
[9] M. Caminada. "Semi-Stable Semantics". In: *COMMA*. 2006, pp. 121–130.
[10] Y. Charalabidis, R. J. Gonalves, and K. Popplewell. "Developing a Science Base for Enterprise Interoperability". In: *Enterprise Interoperability IV*. 2010, pp. 245–254.
[11] G. Charwat, J. P. Wallner, and S. Woltran. "Utilizing ASP for Generating and Visualizing Argumentation Frameworks". In: *ASPOCP*. 2012, pp. 51–65.
[12] S. Doutre and J. Mengin. "On Sceptical Versus Credulous Acceptance for Abstract Argument Systems". In: *JELIA*. 2004, pp. 462–473.
[13] F. Duchateau, Z. Bellahsene, and R. Coletta. "Matching and Alignment: What Is the Cost of User Post-Match Effort?" In: *OTM*. 2011, pp. 421–428.
[14] F. Duchateau et al. "(Not) yet another matcher". In: *CIKM*. 2009, pp. 1537–1540.
[15] P. M. Dung, P. Mancarella, and F. Toni. "Computing ideal sceptical argumentation". In: *Artif. Intell.* (2007), pp. 642–674.
[16] P. M. Dung. "On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games". In: *Artif. Intell.* 77.2 (1995), pp. 321–358.
[17] B. Elvesaeter et al. "Towards enterprise interoperability service utilities". In: *ECOCW*. 2008, pp. 224–229.
[18] C. D. Emele, T. J. Norman, and S. Parsons. "Argumentation strategies for plan resourcing". In: *AAMAS*. 2011, pp. 913–920.
[19] P. Faratin, C. Sierra, and N. R. Jennings. "Negotiation decision functions for autonomous agents". In: *RAS* (1998), pp. 159 –182.
[20] A. Gal. *Uncertain Schema Matching*. Morgan & Claypool Publishers, 2011.

[6] `http://www.sap.com`
[7] `http://www.momentumni.org`

[21] A. Gal and T. Sagi. “Tuning the ensemble selection process of schema matchers”. In: *JIS* (2010), pp. 845–859.
[22] A. Gal et al. “Completeness and Ambiguity of Schema Cover”. In: *CoopIS*. 2013.
[23] A. Gal et al. “Making sense of top-k matchings: a unified match graph for schema matching”. In: *IIWeb*. 2012, 6:1–6:6.
[24] K. Grosse, C. I. Chesevar, and A. G. Maguitman. “An Argument-based Approach to Mining Opinions from Twitter.” In: *AT*. 2012, pp. 408–422.
[25] P. Haas et al. “Data is Dead Without What-If Models”. In: *PVLDB*. 2011, pp. 11–14.
[26] S. Heras et al. “The Role of Argumentation on the Future Internet: Reaching agreements on Clouds”. In: *AT*. 2012, pp. 393–407.
[27] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. “Pay-as-you-go user feedback for dataspace systems”. In: *SIGMOD*. 2008, pp. 847–860.
[28] R. Kowalczyk and V. Bui. “On Constraint-Based Reasoning in e-Negotiation Agents”. In: *AMEC*. 2001, pp. 31–46.
[29] S. Kraus, K. Sycara, and A. Evenchik. “Reaching agreements through argumentation: a logical model and implementation”. In: *AI* (1998), pp. 1–69.
[30] Y. Lee et al. “eTuner: tuning schema matching software using synthetic scenarios”. In: *JVLDB* (2007), pp. 97–122.
[31] M. Nagarajan et al. “Semantic interoperability of web services-challenges and experiences”. In: *ICWS*. 2006, pp. 373–382.
[32] Q. V. H. Nguyen et al. “An MAS Negotiation Support Tool for Schema Matching (Demonstration)”. In: *AAMAS*. 2013, pp. 1391–1392.
[33] Q. V. H. Nguyen et al. “Minimizing Human Effort in Reconciling Match Networks”. In: *ER*. 2013.
[34] Q. V. H. Nguyen et al. “On Leveraging Crowdsourcing Techniques for Schema Matching Networks”. In: *DASFAA*. 2013, pp. 139–154.
[35] A. M. Ouksel and A. Sheth. “Semantic interoperability in global information systems”. In: *SIGMOD* (1999), pp. 5–12.
[36] S. Parsons, C. Sierra, and N. Jennings. “Agents that Reason and Negotiate by Arguing”. In: *JLC* (1998), pp. 261–292.
[37] E. Peukert, J. Eberius, and E. Rahm. “AMC - A framework for modelling and comparing matching systems as matching processes”. In: *ICDE*. 2011, pp. 1304–1307.
[38] H. Prakken. “Some Reflections on Two Current Trends in Formal Argumentation”. In: *Logic Programs, Norms and Action*. 2012, pp. 249–272.
[39] Y. Qi, K. S. Candan, and M. L. Sapino. “FICSR: feedback-based inconsistency resolution and query processing on misaligned data sources”. In: *SIGMOD*. 2007, pp. 151–162.
[40] E. Rahm and P. A. Bernstein. “A Survey of Approaches to Automatic Schema Matching”. In: *JVLDB* (2001), pp. 334–350.
[41] I. Rahwan, F. Zablith, and C. Reed. “Towards large scale argumentation support on the semantic web”. In: *AAAI*. 2007, pp. 1446–1451.
[42] I. Rahwan et al. “Argumentation-based negotiation”. In: *KER* (2003), pp. 343–375.
[43] S. Sá and J. a. Alcântara. “Cooperative dialogues with conditional arguments”. In: *AAMAS*. 2012, pp. 501–508.
[44] K. Saleem and Z. Bellahsene. “Complex Schema Match Discovery and Validation through Collaboration”. In: *OTM*. 2009, pp. 406–413.
[45] T. Sandholm. “Algorithm for optimal winner determination in combinatorial auctions”. In: *AI* (2002), pp. 1–54.
[46] C. Sierra et al. “A Framework for Argumentation-Based Negotiation”. In: *ATAL*. 1997, pp. 167–182.
[47] K. P. Smith et al. “The Role of Schema Matching in Large Enterprises”. In: *CIDR*. 2009.